STRUKTUR DATA NON LINEAR

"BINARY SEARCH TREE"

MODUL PERTEMUAN KE – 4 : DELETE 0 & 1 ANAK

DISUSUN OLEH :

Nama : Andreas Nathanael Priambodo

NIM : 215314043

TEKNIK INFORMATIKA FAKULTAS SAINS DAN TEKNOLOGI UNIVERSITAS SANATA DHARMA YOGYAKARTA 2023

A. Soal

TreeNode	
- data	: int
- leftNode	: TreeNode
- rightNode	: TreeNode
- parent	: TreeNode
+ TreeNode (int, TreeNode)	: Constuctor
+ insert(int)	: void

Tree	
– root	: TreeNode
+ Tree()	: Constructor
+ insertNode(int)	: void
+ delete(int)	: boolean
+ getCurrent(int)	: TreeNode

1. Buatlah program Tree (pohon biner) berdasarkan diagram kelas UML diatas.
2. Buatlah algoritma untuk method insert (rekusif), insertNode dan delete.
3. Lengkapi juga dengan method main dan eksekusilah program Tree dengan memasukan data secara urut mulai dari 42, 21, 38, 27, 71, 82, 55, 63, 6, 2, 40 dan 12.
4. Lakukan penghapusan pohon mulai dari data 12, 27, 6 dan 55 selanjutnya catat perubahannya.

B. Algoritma

- Insert (class Tree)
  1. Jika root belum ada, maka set node baru sebagai root.
  2. Jika root sudah ada, panggil method insert pada root dan teruskan nilai yang akan diinsert.
- Insert (class TreeNode)
  1. Jika nilai yang akan diinsert kurang dari atau sama dengan nilai node saat ini, lakukan langkah 2. Jika tidak, lakukan langkah 3.
  2. Jika left node dari node saat ini kosong, buat node baru dengan nilai yang diinsert dan jadikan left node dari node saat ini. Jika tidak, panggil method insert pada left node dan teruskan nilai yang akan diinsert.

3. Jika right node dari node saat ini kosong, buat node baru dengan nilai yang diinsert dan jadikan right node dari node saat ini. Jika tidak, panggil method insert pada right node dan teruskan nilai yang akan diinsert.

- Delete
  1. Cek apakah root node tidak null.
  2. Jika ya, panggil method deleteHelper dengan parameter root node dan nilai data yang ingin dihapus.
  3. Jika tidak, kembalikan false.
- deleteHelper
  1. Jika node adalah null, kembalikan null.
  2. Jika nilai data sama dengan node.getData(), maka cek apakah node tidak memiliki left node dan right node. Jika ya, kembalikan null karena node tersebut merupakan leaf node. Jika tidak, cek apakah node hanya memiliki left node. Jika ya, kembalikan left node. Jika tidak, cek apakah node hanya memiliki right node. Jika ya, kembalikan right node.
  3. Jika node bukan leaf node dan tidak memiliki left node atau right node, maka tampilkan pesan "Tidak berhasil menghapus" dan kembalikan node tersebut.
  4. Jika nilai data yang ingin dihapus lebih kecil dari node.getData(), maka panggil deleteHelper dengan parameter node.getLeftNode() dan data yang ingin dihapus.
  5. Jika tidak, panggil deleteHelper dengan parameter node.getRightNode() dan data yang ingin dihapus.
  6. Kembalikan node.

C. Program

- MainTree

```
package binary_search_tree_copy1;

public class Binary_Search_tree {

   public static void main(String[] args) {
      Tree Ob = new Tree();
      Ob.Insert(42);
```

```
        Ob.Insert(21);
        Ob.Insert(38);
        Ob.Insert(27);
        Ob.Insert(71);
        Ob.Insert(82);
        Ob.Insert(55);
        Ob.Insert(63);
        Ob.Insert(6);
        Ob.Insert(2);
        Ob.Insert(40);
        Ob.Insert(12);
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
        if (Ob.delete(0)) {
            System.out.println("Berhasil Delete");
        } else {
            System.out.println("Tidak Berhasil Delete");
        }
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
    }

}
```

- Tree

```
package binary_search_tree_copy1;


public class Tree {
```

```
private TreeNode root;

public Tree() {
    root = null;
}

public Tree(TreeNode root) {
    this.root = root;
}

public TreeNode getRoot() {
    return root;
}

public void setRoot(TreeNode root) {
    this.root = root;
}

public void preOrderTransversal() {
    preOrderHelper(root);
}

public void inOrderTransversal() {
    inOrderHelper(root);
}

public void postOrderTransversal() {
    postOrderHelper(root);
}

public void preOrderHelper(TreeNode node) {
    if (node != null) {
        System.out.print(node.getData() + " ");
```

```
        preOrderHelper(node.getLeftNode());
        preOrderHelper(node.getRightNode());
    }

}

public void inOrderHelper(TreeNode node) {
    if (node != null) {
        inOrderHelper(node.getLeftNode());
        System.out.print(node.getData() + " ");
        inOrderHelper(node.getRightNode());
    }

}

public void postOrderHelper(TreeNode node) {
    if (node != null) {
        postOrderHelper(node.getLeftNode());
        postOrderHelper(node.getRightNode());
        System.out.print(node.getData() + " ");
    }

}

public void Insert(int in) {
    if (root == null) {
        root = new TreeNode(in);
        System.out.println("Root " + root.getData());
    } else {
        root.insert(in);
    }
}
```

```
public TreeNode getCurrent(int current) {
    TreeNode data = root;
    while (data != null) {
        if (current == data.getData()) {
            return data;
        } else if (current < data.getData()) {
            data = data.getLeftNode();
        } else {
            data = data.getRightNode();
        }
    }
    return null;
}

public TreeNode search(int search) {
    return getCurrent(search);
}

public boolean delete(int data) {
    if (root != null) {
        root = deleteHelper(root, data);
        return true;
    }
        return false;
}

private TreeNode deleteHelper(TreeNode node, int data) {
    if (node == null) {
        return null;
    }

    if (data == node.getData()) {
        if (node.getLeftNode() == null && node.getRightNode() == null) {
```

```
            return null;
        } else if (node.getLeftNode() == null) {
            return node.getRightNode();
        } else if (node.getRightNode() == null) {
            return node.getLeftNode();
        } else {
            System.out.println("Tidak berhasil menghapus");
            return node;
        }
    } else if (data < node.getData()) {
        node.setLeftNode(deleteHelper(node.getLeftNode(), data));
    } else {
        node.setRightNode(deleteHelper(node.getRightNode(), data));
    }
    return node;
  }
}
```

- TreeNode

```
package binary_search_tree_copy1;

public class TreeNode {

  private int data;
  private TreeNode leftNode;
  private TreeNode rightNode;
  private TreeNode parent;

  public TreeNode(int data) {
    this.data = data;
    leftNode = rightNode = parent = null;
  }
```

```
public TreeNode(int data, TreeNode parent) {
    this.data = data;
    leftNode = rightNode = null;
    this.parent = parent;
}

public int getData() {
    return data;
}

public void setData(int data) {
    this.data = data;
}

public TreeNode getLeftNode() {
    return leftNode;
}

public void setLeftNode(TreeNode leftNode) {
    this.leftNode = leftNode;
    if (leftNode != null) {
        leftNode.setParent(this);
    }
}

public TreeNode getRightNode() {
    return rightNode;
}

public void setRightNode(TreeNode rightNode) {
    this.rightNode = rightNode;
    if (rightNode != null) {
        rightNode.setParent(this);
```

```
        }
    }

    public TreeNode getParent() {
        return parent;
    }

    public void setParent(TreeNode parent) {
        this.parent = parent;
    }

    public void insert(int in) {
        if (in <= data) {
            if (leftNode == null) {
                System.out.println(in + " Left "+this.getData());
                setLeftNode(new TreeNode(in, this));
            } else {
                leftNode.insert(in);
            }
        } else {
            if (rightNode == null) {
                System.out.println(in + " Right "+this.getData());
                setRightNode(new TreeNode(in, this));
            } else {
                rightNode.insert(in);
            }
        }
    }
}
```

D. Output

Code Delete 12, 27, 6, 55

```
package binary_search_tree_copy1;

public class Binary_Search_tree {

    public static void main(String[] args) {
        Tree Ob = new Tree();
        Ob.Insert(42);
        Ob.Insert(21);
        Ob.Insert(38);
        Ob.Insert(27);
        Ob.Insert(71);
        Ob.Insert(82);
        Ob.Insert(55);
        Ob.Insert(63);
        Ob.Insert(6);
        Ob.Insert(2);
        Ob.Insert(40);
        Ob.Insert(12);
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
        if (Ob.delete(12)) {
            System.out.println("Berhasil Delete");
        } else {
            System.out.println("Tidak Berhasil Delete");
        }
        Ob.delete(27);
        System.out.println();
        System.out.println("PreOrder : ");
        Ob.preOrderTransversal();
        System.out.println();
        Ob.delete(6);
```

```
                System.out.println();
                System.out.println("PreOrder : ");
                Ob.preOrderTransversal();
                System.out.println();
                Ob.delete(55);
                System.out.println();
                System.out.println("PreOrder : ");
                Ob.preOrderTransversal();
                System.out.println();
            }
}
```

```
run:
Root 42
21 Left 42
38 Right 21
27 Left 38
71 Right 42
82 Right 71
55 Left 71
63 Right 55
6 Left 21
2 Left 6
40 Right 38
12 Right 6

PreOrder :
42 21 6 2 12 38 27 40 71 55 63 82
Berhasil Delete

PreOrder :
42 21 6 2 38 40 71 55 63 82

PreOrder :
42 21 2 38 40 71 55 63 82

PreOrder :
42 21 2 38 40 71 63 82
BUILD SUCCESSFUL (total time: 0 seconds)
```

E. Analisa

- MainTree

- Tree

  Didalam class tree ini ada attribute yang Bernama root yang berbentuk private yang dimana root ini bertipekan data class TreeNode lalu dibawahnya terdapat constructor yang digunakan untuk pendeklarasian root ini null dan dibawahnya lagi ada kontruktor lagi yang akan terisi apabila akan menambah TreeNode yang sudah ada di main kelas sehingga nanti dapat menggunakan root ini selanjutnya ada setter dan getter dari root dan juga ada method insert dan juga method search untuk inster sendiri berisikan parameter integer Bernama in dan untuk search Sendiri juga menggunakan integer juga tetapi Bernama cari untuk Search akan melakukan algoritma dari Search dan untuk Inser juga akan melakukan algoritma dari Search. Ada tahmbahan untuk sekarang terdapat pre order, in order, dan post order transversal method yang nanti dia memanggil method lain untuk method helper dari masing masing order. Yang pertama untuk preOrderHelper yang berisikan parameter node yang bertipekan class TreeNode didalam bodynya mengecek apakah node nya kosong dan jika kosong akan langsung dikembalikan ke method atasnya yaitu untuk transversal lalu akan Kembali lagi ke main classnya yaitu MainTree selanjutnya untuk bawahnya lagi ada print untuk mencetak data node nya lalu dibawah ada pemanggilan method lagi dan dia memanggil dirinya sendiri atau istilahnya dengan cara rekursif dia akan berulang kali mencari untuk melihat apakah ada leftNode atau tidak dan djika tidak akan Kembali ke rekursif awalan tadi begitu seterusnya. Sama seperti sebelumnya untuk inOrder dengan postOrder yang berbeda hanya dalam urutan print dari node nya karena print disini bisa dianggap sebagai root nya yang akan langsung dicetak. Ada tambahan yaitu untuk Method delete() memiliki parameter data yang merupakan data yang akan dihapus. Pada awal method, dilakukan pengecekan apakah root dari tree-nya tidak null. Jika tidak null, method akan memanggil method deleteHelper() untuk menghapus node yang mengandung data tersebut. Ada juga tambahan Method deleteHelper() merupakan method rekursif yang memproses setiap node dalam tree untuk mencari node yang mengandung data yang akan dihapus. Jika node ditemukan, dilakukan pengecekan terhadap node tersebut, apakah memiliki satu anak atau tidak memiliki anak. Jika tidak memiliki anak, node tersebut dihapus dari tree. Jika hanya memiliki satu anak, anak tersebut akan menjadi pengganti node tersebut di tree. Jika memiliki dua anak, maka node tersebut tidak dapat dihapus.

Jika data yang dicari lebih kecil dari data pada node saat ini, method akan dipanggil lagi pada anak kiri dari node tersebut. Jika lebih besar, maka method akan dipanggil pada anak kanan. Setelah selesai memproses setiap node, method akan mengembalikan node yang sudah diproses. Method delete() memiliki parameter data yang merupakan data yang akan dihapus. Pada awal method, dilakukan pengecekan apakah root dari tree-nya tidak null. Jika tidak null, method akan memanggil method deleteHelper() untuk menghapus node yang mengandung data tersebut. Method deleteHelper() merupakan method rekursif yang memproses setiap node dalam tree untuk mencari node yang mengandung data yang akan dihapus. Jika node ditemukan, dilakukan pengecekan terhadap node tersebut, apakah memiliki satu anak atau tidak memiliki anak. Jika tidak memiliki anak, node tersebut dihapus dari tree. Jika hanya memiliki satu anak, anak tersebut akan menjadi pengganti node tersebut di tree. Jika memiliki dua anak, maka node tersebut tidak dapat dihapus. Jika data yang dicari lebih kecil dari data pada node saat ini, method akan dipanggil lagi pada anak kiri dari node tersebut. Jika lebih besar, maka method akan dipanggil pada anak kanan. Setelah selesai memproses setiap node, method akan mengembalikan node yang sudah diproses. Method insert pada class Tree digunakan untuk memasukkan sebuah node baru ke dalam binary search tree. Jika tree masih kosong (root node belum ada), maka method akan membuat root node baru dengan nilai yang diinput. Jika tree sudah terisi, maka method akan memanggil method insert pada class TreeNode dengan parameter nilai yang diinput, dimulai dari root node.

- TreeNode

  Didalam TreeNode ini berisikan 3 attribut untuk data yang bertipekan integer lalu ada leftNode dan rightNode yang bertipekan TreeNode lalu ada 1 buah contructor yang berparameter data unguk mengisikan attribute data dan juga ada penginisialisasian untuk leftNode dan juga rightNode menjadi null selanjutnya ada setter dan getter untuk data dan juga leftNode dan rightNode. Method insert pada class TreeNode digunakan untuk memasukkan sebuah node baru ke dalam binary search tree. Jika nilai yang diinput lebih kecil atau sama dengan nilai pada node saat ini, maka method akan mencoba memasukkan nilai tersebut ke sub-tree kiri dari node saat ini. Jika sub-tree kiri belum terisi, maka method akan membuat node baru pada sub-tree kiri tersebut dengan nilai yang diinput. Jika

sub-tree kiri sudah terisi, maka method akan memanggil method insert pada sub-tree kiri tersebut dengan parameter nilai yang diinput. Jika nilai yang diinput lebih besar dari nilai pada node saat ini, maka method akan mencoba memasukkan nilai tersebut ke sub-tree kanan dari node saat ini. Jika sub-tree kanan belum terisi, maka method akan membuat node baru pada sub-tree kanan tersebut dengan nilai yang diinput. Jika sub-tree kanan sudah terisi, maka method akan memanggil method insert pada sub-tree kanan tersebut dengan parameter nilai yang diinput. Pada setiap node baru yang berhasil dibuat, method setParent juga dipanggil untuk menunjukkan node yang menjadi parent dari node tersebut.